

UNIVERSITATEA "ALEXANDRU-IOAN CUZA" DIN IAȘI

**FACULTATEA DE INFORMATICĂ**



LUCRARE DE LICENȚĂ

**Sistem de pontaj și acces**

propusă de

**Bordei Mihai-Gabi**

**Sesiunea: iunie, 2023**

Coordonator științific

**Lect. Dr. Bîrjoveanu Cătalin**

UNIVERSITATEA "ALEXANDRU-IOAN CUZA" DIN IAȘI

**FACULTATEA DE INFORMATICĂ**


## **Sistem de pontaj și acces**

**Bordei Mihai-Gabi**

**Sesiunea: iunie, 2023**

Coordonator științific

**Lect. Dr. Bîrjoveanu Cătălin**

Avizat,  
Îndrumător lucrare de licență,  
Lect. Dr. Bîrjoveanu Cătălin.  
Semnătura: 

Data: 22.06.2023 .....

### Declarație privind originalitatea conținutului lucrării de licență

Subsemnatul **Mihai-Gabi Bordei** domiciliat în **România, jud. Vaslui, mun. Vaslui, strada Ștefan cel Mare, bl. 151 b, et. 1, ap. 13**, născut la data de **08 noiembrie 2001**, identificat prin CNP **5011108374516**, absolvent al Facultății de informatică, **Facultatea de informatică** specializarea **informatică**, promoția 2023, declar pe propria răspundere cunoscând consecințele falsului în declarații în sensul art. 326 din Noul Cod Penal și dispozițiile Legii Educației Naționale nr. 1/2011 art. 143 al. 4 și 5 referitoare la plagiat, că lucrarea de licență cu titlul **Sistem de pontaj și acces** elaborată sub îndrumarea domnului **Lect. Dr. Bîrjoveanu Cătălin**, pe care urmează să o susțin în fața comisiei este originală, îmi aparține și îmi asum conținutul său în întregime.

De asemenea, declar că sunt de acord ca lucrarea mea de licență să fie verificată prin orice modalitate legală pentru confirmarea originalității, consimțind inclusiv la introducerea conținutului ei într-o bază de date în acest scop.

Am luat la cunoștință despre faptul că este interzisă comercializarea de lucrări științifice în vederea facilitării falsificării de către cumpărător a calității de autor al unei lucrări de licență, de diplomă sau de disertație și în acest sens, declar pe proprie răspundere că lucrarea de față nu a fost copiată ci reprezintă rodul cercetării pe care am întreprins-o.

Data: 22.06.2023 .....

Semnătura:  .....

## Declarație de consimțământ

Prin prezenta declar că sunt de acord ca lucrarea de licență cu titlul **Sistem de pontaj și acces**, codul sursă al programelor și celelalte conținuturi (grafice, multimedia, date de test, etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de informatică.

De asemenea, sunt de acord ca Facultatea de informatică de la Universitatea "Alexandru-Ioan Cuza" din Iași, să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Absolvent **Bordei Mihai-Gabi**

Data: 22.06.2023 .....

Semnătura: ..... 

# Cuprins

|   |           |
|---|-----------|
| <b>Motivație</b>  | <b>2</b>  |
| <b>Introducere</b>  | <b>3</b>  |
| <b>1 Descriere generală</b>   | <b>4</b>  |
| 1.1 Prezentarea problemei . . . . .                                     | 5         |
| 1.2 Aspecte relevante . . . . .   | 6         |
| 1.2.1 Simularea cardului . . . . .                                      | 7         |
| 1.2.2 Contorizarea timpului . . . . .                                   | 7         |
| 1.3 Mediu de lucru . . . . .  | 8         |
| <b>2 Descrierea soluției</b>  | <b>9</b>  |
| 2.1 Uneltele software folosite . . . . .                                | 9         |
| 2.2 Descrierea implementării . . . . .                                  | 10        |
| 2.3 Java Card Applet . . . . .  | 12        |
| 2.3.1 Structura applet-ului . . . . .                                   | 13        |
| 2.3.2 Comenziile suportate de applet . . . . .                          | 15        |
| 2.3.3 Metode din cadrul applet-ului . . . . .                           | 16        |
| 2.3.4 Conceptul de comandă APDU . . . . .                               | 19        |
| 2.3.5 Script-uri de comenzi APDU . . . . .                              | 20        |
| 2.4 Aplicația Terminal . . . . .  | 21        |
| 2.4.1 Pachetele care fac legătura cu Java Card Applet si baza de date . | 22        |
| 2.4.2 Pachetul care face legătura cu utilizatorul . . . . .             | 27        |
| 2.5 Baza de date . . . . .  | 30        |
| 2.6 Probleme întâmpinate și soluțiile acestora . . . . .                | 31        |
| <b>3 Funcționalități și avantaje ale cardurilor inteligente</b>         | <b>33</b> |
| 3.1 Funcționalități . . . . .   | 33        |

|                     |                                    |           |
|---------------------|------------------------------------|-----------|
| 3.1.1               | Accesul în diferite zone . . . . . | 33        |
| 3.1.2               | Pontajul orelor de lucru . . . . . | 34        |
| 3.2                 | Avantaje . . . . .                 | 35        |
| <b>Concluzii</b>    |                                    | <b>36</b> |
| <b>Bibliografie</b> |                                    | <b>37</b> |

# Motivație

Am ales această temă pentru a evidenția avantajele utilizării **cardurilor inteligente**<sup>1</sup> în gestionarea pontajului orelor de lucru și a accesului în anumite zone. Prin implementarea acestor dispozitive inteligente, echipate cu cipuri electronice, putem obține o soluție eficientă și securizată. Alegerea unei soluției cu carduri inteligente pune în valoare multiple avantaje, cum ar fi ușurința în folosire, precizia și transparența înregistrărilor, precum și managementul simplu al accesului. Prin utilizarea cardurilor inteligente, putem realiza un management eficient al resurselor, reducând erorile umane care pot apărea în acest context. Cardurile inteligente reprezintă o soluție modernă și fiabilă, adaptabilă în diverse domenii de activitate, care oferă avantajele necesare pentru a optimiza activitățile comune de la nivelul unei organizații și care asigură un mediu sigur și eficient.

---

<sup>1</sup>Un card inteligent este un dispozitiv care conține un cip electronic integrat și poate stoca și procesa informații

# Introducere

Prin această aplicație, am dorit să abordez o problemă contemporană și să găesc o soluție eficientă și ușor de implementat prin intermediul cardurilor inteligente. Am observat că gestionarea pontajului și a accesului în diverse zone reprezintă uneori o provocare pentru organizații, fiind nevoie de un sistem precis și sigur. Prin utilizarea cardurilor inteligente, am identificat o modalitate modernă și eficientă de a rezolva această problemă, oferind o soluție care poate fi implementată cu ușurință și care aduce beneficii semnificative. Utilizarea cardurilor inteligente în cadrul problematicii propuse de mine, gestionarea pontajului și a accesului, aduce multiple avantaje. Prin simpla scanare a cardului într-un terminal, sistemul automatizează procesele de înregistrare a orelor de muncă și de control al accesului, eliminând nevoia înregistrărilor manuale și reducând riscul de erori umane sau discrepante în evidența orelor lucrate.

Cardurile inteligente asigură un nivel sporit de securitate, deoarece fiecare card este înregistrat în sistem și poate fi personalizat cu privilegii specifice pentru fiecare angajat. Astfel, doar persoanele autorizate pot accesa anumite zone restricționate, contribuind la reducerea riscului de acces neautorizat și sporind siguranța organizației.

Aceste carduri sunt ușor de utilizat de către angajați, fiind similare cu un card bancar. Interacțiunea simplă și intuitivă cu sistemul facilitează utilizarea lor fără necesitatea instruirii suplimentare. Pe termen lung, sistemul bazat pe carduri inteligente este scalabil și adaptabil la nevoile organizației. Utilizarea cardurilor inteligente în aplicația mea de gestionare a pontajului și accesului în zone restricționate aduce eficiență, securitate și ușurință în utilizare.



# Capitolul 1

## Descriere generală

Acest capitol are scopul de a oferi o prezentare detaliată a sistemului propus de mine pentru pontajul orelor de muncă și controlul accesului în cadrul unei organizații. Voi furniza informații relevante despre modul în care sistemul va funcționa și cum va gestiona orele de muncă în contextul specific al unei companii sau organizații. Voi oferi informații despre modul în care sistemul propus va optimiza procesul de pontaj al orelor de muncă, asigurând înregistrări precise și eficiente. Voi descrie modul în care cardurile inteligente vor fi utilizate pentru a înregistra intrările și ieșirile angajaților, facilitând procesul de înregistrare și monitorizare a timpului de lucru.

Prin intermediul acestui capitol, voi încerca să ofer o înțelegere detaliată despre modul în care sistemul propus va contribui la eficiența și transparența procesului de pontaj al orelor de muncă în cadrul unei companii.

Tot în acest capitol, voi descrie câteva aspecte relevante ale aplicației realizate de mine, cu scopul de a oferi o înțelegere mai bună asupra conceptului de sistem de pontaj și acces bazat pe utilizarea cardurilor inteligente.

Astfel, acest capitol evidențiază modul în care funcționează sistemul propus și câteva aspect relevante despre aplicație. Aș vrea să subliniez faptul că, în acest capitol, voi discuta mai mult despre descrierea funcționalităților aplicației și mai puțin despre partea tehnică. O descriere completă din punct de vedere tehnic va fi prezentată în capitolul următor.

## 1.1 Prezentarea problemei

Problematica abordată pentru exemplificarea avantajelor folosirii tehnologiilor bazate pe carduri inteligente este reprezentată de simularea unui mediu dintr-o companie, în care se dorește utilizarea unui card pentru a putea monitoriza orele de lucru ale angajaților și accesul acestora în diferite zone ale companiei.

Fiecare card de acces va conține informații despre angajatul atribuit acestui card, precum ID-ul angajatului, zonele în care are permisiunea de acces, numărul de ore și minute lucrate în ziua curentă și numărul de ore și minute lucrate în luna curentă. ID-ul angajatului trebuie să fie un număr întreg unic. Acest ID va fi folosit și pentru memorarea informațiilor despre angajații la nivelul unei baze de date.

Pentru implementarea acestui sistem, se folosește o *aplicație Terminal* realizată în **Java**, care comunică cu o aplicație *Java Card Applet* prin intermediul pachetului `com.sun.javacard.apduio`. *Aplicația Terminal* interacționează cu o bază de date de tipul **Oracle**, în care sunt stocate informații precum ID-ul angajatului, numele, numărul de ore lucrate în fiecare zi din luna respectivă și codul PIN al angajatului.

Aplicația rezolvă următoarele scenarii de utilizare:

1. La intrarea în companie, angajatul introduce PIN-ul în *aplicația Terminal*, care trimite acest cod către *Java Card Applet* pentru validare. Dacă PIN-ul este validat, angajatului i se permite intrarea în companie, se începe înregistrarea orelor pentru ziua respectivă și *aplicația Terminal* afișează mesajul Bine ați venit! `<Nume HH:MM:DD:LL:AA>`.
2. La fiecare intrare într-o zonă, angajatul trebuie să prezinte cardul. Dacă PIN-ul este validat și angajatul are permisiunea de acces în zona respectivă (verificat prin comunicarea între *aplicația Terminal* și *Java Card Applet*), *aplicația Terminal* afișează `Access permis <Număr_zonă> <Nume HH:MM:DD:LL:AA>`, în caz contrar afișează mesajul `Access Interzis <Număr_zonă>`.
3. La fiecare ieșire dintr-o zonă, angajatul trebuie să prezinte cardul. Prin intermediul *aplicației Terminal* se afișează mesajul următor: `Ieșire din zona <Număr_zonă> <Nume HH:MM:DD:LL:AA>`. Angajatul poate ieși dintr-o zonă doar dacă anterior a intrat în acea zonă.
4. La ieșirea din companie, angajatul trebuie să prezinte cardul. Aplicația afișează mesajul următor: `La revedere! Ai lucrat <HH:MM> ore azi, actuali-`

zează numărul de ore lucrate în ziua respectivă și înregistrează orele în baza de date.

5. Prin intermediul *aplicației Terminal*, angajații pot verifica numărul de ore lucrate în companie în luna respectivă. Această informație este importantă pentru angajați.

(a) Dacă angajatul a acumulat deja mai mult de 160 de ore lucrate în luna respectivă, atunci când încearcă să intre în companie data viitoare, *aplicația Terminal* va afișa un mesaj de tipul `Ati efectuat deja numarul de ore necesar luna aceasta.`

(b) Dacă numărul de ore efectuat în luna respectivă este de minim 152 de ore, *aplicația Terminal* afișează un mesaj de tipul `Mai aveti de recuperat <x> ore (la plecarea din companie în ultima zi a lunii).`

(c) Dacă numărul de ore efectuat în luna respectivă este mai mic de 152 de ore, *aplicația Terminal* afișează mesajul `Primiti o penalizare de 10% (la plecarea din companie în ultima zi a lunii).`

6. Dacă angajatul se află în companie, acesta poate să-și schimbe codul PIN. Noul cod PIN este transmis *aplicației Terminal*, care îl transmite mai departe la *Java Card Applet*. Acesta verifică dacă noul cod PIN este valid, având minim 4 cifre și maxim 8. După verificare, dacă noul cod PIN este valid, atunci se actualizează PIN-ul în baza de date și se cere reverificarea codului PIN pentru orice acțiune din cadrul companiei. În caz contrar, dacă noul cod PIN nu este valid, atunci se afișează mesaje specifice de eroare în funcție de tipul erorii.

## 1.2 Aspecte relevante

În această secțiune, voi evidenția câteva aspecte relevante ale aplicației dezvoltate pentru gestionarea pontajului și accesului în diverse zone în cadrul unei companii. Prin implementarea acestor funcționalități esențiale, am urmărit optimizarea proceselor de monitorizare a orelor de muncă și controlul accesului, aducând beneficii semnificative atât pentru angajat, cât și pentru companie. Voi prezenta în detaliu aceste aspecte cheie, care contribuie la eficiența și securitatea sistemului.

### 1.2.1 Simularea cardului

Un aspect important este reprezentat de simularea atingerii cardului pe un terminal fizic, în cadrul aplicației. Pentru a realiza această simulare, voi utiliza o **instanță cref**<sup>1</sup> a cardului și un server care să gestioneze comunicarea. În codul de implementare, se utilizează următoarele instrucțiuni:

```
String crefFilePath = "c:\\Program Files(x86)\\Oracle  
\\Java Card Development Kit Simulator 3.1.0\\bin\\cref.bat";  
Process process;  
process = Runtime.getRuntime().exec(crefFilePath);
```

Aceste linii de cod permit lansarea fișierului `cref.bat` care se ocupă de modul în care are loc simularea cardul. Prin intermediul acestui proces, am putut obține funcționalitatea necesară pentru a simula atingerea cardului pe un terminal fizic și pentru a efectua diverse operații în cadrul aplicației.

Acest aspect este deosebit de important în dezvoltarea sistemului, deoarece mi-a permis să testez și să validez comportamentul sistemului într-un mediu virtual înainte de implementarea sa într-un mediu real.

Un alt aspect important legat de simularea cardului este reprezentat de faptul că fiecare angajat care are informațiile stocate în baza de date are cardul său. Deci, pentru fiecare angajat din baza de date se va crea un card, așa cum am descris mai sus.

În capitolul următor, voi oferi mai multe detalii tehnice despre aplicația mea, explorând în profunzime aspectele de implementare și funcționalitate.

### 1.2.2 Contorizarea timpului

Un alt aspect important al sistemului propus este reprezentat de contorizarea timpului. În cadrul aplicației, timpul petrecut de un angajat în cadrul companiei este exclusiv contorizat în cadrul aplicației *Java Card Applet*. Inițial, când are loc crearea cardului, sunt preluate din baza de date orele și minutele lucrate în ziua curentă și apoi, odată cu intrarea în companie, adică verificarea codului PIN, începe contorizarea timpului.

---

<sup>1</sup>O instanță "cref" este o instanță a programului "cref.bat" care face parte din Java Card Development Kit. Această instanță permite simularea și testarea aplicațiilor Java Card pe un mediu de dezvoltare local înainte de a fi încărcate pe un card real.

La final, la ieșirea din companie, se oprește contorizarea și se trimit noile date despre ore și minute la aplicația *Terminal* cu scopul de a actualiza informațiile din baza de date. Mai multe informații tehnice vor fi prezentate în capitolul următor.

## 1.3 Mediu de lucru

În cadrul dezvoltării aplicației, am utilizat mediul de lucru **Eclipse** în combinație cu **Java Card Development Kit 3.1**. Această combinație m-a ajutat să dezvolt și să testez aplicația *Java Card Applet* într-un mod eficient și rapid din punctul de vedere al timpului.

**Java Card Development KIT 3.1** reprezintă un set de instrumente și biblioteci, concepute pentru a ușura dezvoltarea aplicațiilor **Java Card**. După instalarea *kit*-ului de dezvoltare, am avut acces la diferite unelte pentru compilarea și gestionarea **applet-urilor**<sup>2</sup> **Java Card**. Un aspect cheie al **Java Card Development Kit 3.1** este reprezentat de emulatorul de carduri, care m-a ajutat să testez și să fac *debugging* asupra aplicației mele, *Java Card Applet*. Emulatorul de carduri mi-a oferit posibilitatea de a introduce date și comenzi de testare, de a urmări starea cardului, precum și de a observa răspunsurile și comportamentul *applet*-ului. Vezi figura 1.1.

În plus, mediul de lucru **Eclipse** a oferit un set de instrumente potrivite pentru dezvoltarea aplicațiilor **Java Card**. Am avut acces la funcționalități precum completarea automată a codului, navigarea ușoară prin proiecte și fișiere.

Prin urmare, utilizarea mediului de lucru **Eclipse** în combinație cu **Java Card Development Kit 3.1**, am putut să dezvolt și să testez aplicații *Java Card Applet* într-un mod eficient și productiv.

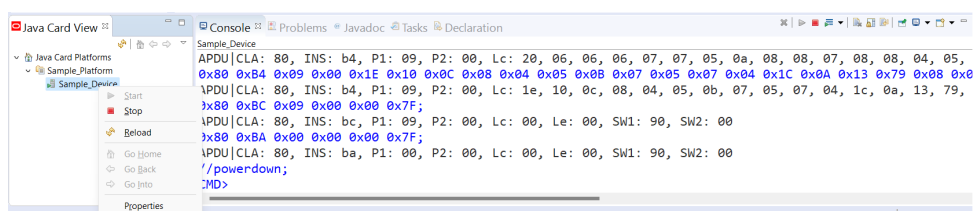


Figura 1.1: Emulatorul Java Card

<sup>2</sup>Un applet este o mică aplicație software care rulează în cadrul unui mediu de execuție mai mare.

# Capitolul 2

## Descrierea soluției

În acest capitol, voi prezenta în detaliu soluția mea pentru problematica descrisă în capitolul precedent. Voi aborda aspecte relevante legate de uneltele *software* folosite, sistemul propus pentru rezolvarea problemei și provocările întâmpinate pe parcursul implementării, împreună cu soluțiile găsite.

### 2.1 Uneltele software folosite

În cadrul acestui subcapitol, voi prezenta uneltele *software* folosite în implementarea soluției. Am realizat o aplicație scrisă în limbajul **Java**, utilizând atât **Java Card**, cât și **JavaFX**. Totuși, dacă aș face un raport al limbajelor de programare folosite, aș putea spune că 95% am folosit limbajul **Java**, iar restul de 5% este reprezentat de alte limbaje, cum ar fi **CSS** și **PL/SQL**.

În dezvoltarea aplicației mele, am utilizat **Java 11**, o versiune versatilă a limbajului de programare **Java**. **Java 11** aduce cu sine o serie de îmbunătățiri și funcționalități noi care mi-au fost de mare ajutor în crearea aplicației. **Java 11** a fost o alegere excelentă pentru dezvoltarea aplicației mele, deoarece mi-a permis să scriu cod mai concis, să modularizez aplicația și să beneficiaz de multe alte avantaje.

**Java Card** a reprezentat baza tehnologică principală pentru dezvoltarea aplicației. Am utilizat **Java Card Development Kit 3.1** și **Java Card Tools 3.1.0**, instrumente care mi-au furnizat un mediu integrat pentru dezvoltarea și testarea *applet*-ului din cadrul aplicației. **Java Card Development Kit 3.1** mi-a oferit o suită de instrumente specializate în a oferi funcționalități pentru crearea, gestionarea și simularea cardurilor.

Pentru partea de interfață grafică, am folosit **JavaFX**, o platformă de dezvoltare

*software* pentru crearea de aplicații *desktop* și *mobile*. **JavaFX** mi-a oferit un set complex de componente și instrumente pentru crearea unei interfețe ușor de folosit de către un utilizator și totodată atractive din punct de vedere vizual. Un avantaj oferit de **JavaFX** a fost multitudinea de variante pe care le oferă pentru a crea o interfață grafică. Pentru a obține o interfață grafică mai plăcută, am integrat un fișier **CSS** în aplicație. Acesta a fost folosit pentru a realiza stilizarea și formatarea elementelor vizuale, oferind o estetică coerentă și atractivă. Fișierul **CSS** a permis personalizarea aspectului componentelor interfeței grafice, cum ar fi culorile, fonturile, dimensiunile și altele, contribuind astfel la crearea unei experiențe vizuale plăcute pentru utilizatori.

Prin combinarea **Java Card** și **JavaFX**, am reușit să dezvolt o aplicație robustă, care să beneficieze atât de avantajele tehnologiilor **Java Card**, cât și de flexibilitatea și aspectul modern oferit de **JavaFX**. Această combinație a permis implementarea eficientă și destul de clară a sistemului propus drept soluție pentru problematica propusă.

Pentru a putea simula cât mai bine mediul dintr-o companie, aveam nevoie de o structură în care să îmi memorez informațiile despre angajați și zilele lor de muncă, și în acest sens am hotărât să folosesc o bază de date de tip **Oracle**, care este una dintre cele mai populare soluții de gestionare a datelor într-un mod relațional. Utilizarea bazei de date **Oracle** a contribuit la eficiența și funcționalitatea sistemului meu.

În ansamblu, combinarea mediului de lucru **Eclipse**, instalarea **Java Card Development Kit 3.1**, utilizarea **Java 11**, integrarea **Java Card** și **JavaFX**, precum și utilizarea bazei de date **Oracle** au contribuit la dezvoltarea unei aplicații puternice, eficiente și cu o interfață grafică atractivă. Această soluție a răspuns cerințelor problemei abordate, facilitând gestionarea accesului în diferite zone și contorizarea timpului în cadrul companiei.

În continuare, voi detalia implementarea și integrarea acestor unelte software în soluția mea, precum și provocările întâmpinate și soluțiile găsite pentru a asigura o funcționalitate optimă și o experiență de utilizare de calitate.

## 2.2 Descrierea implementării

Pentru a începe descrierea implementării aplicației, voi prezenta structura acesteia. Aplicația este compusă dintr-o aplicație *Terminal* scrisă în **Java**, un *Java Card Applet* și o bază de date de tip **Oracle**. Termenul "*terminal*" se referă la interfața prin care un utilizator poate realiza operațiuni cu cardul creat după *applet*-ul din cadrul componen-

tei *Java Card Applet*.

Implementarea aplicației a necesitat o atentă planificare și coordonare între componentele sale, **aplicația Terminal**, **Java Card Applet** și baza de date. S-au folosit tehnologii și biblioteci adecvate pentru a asigura o comunicare eficientă și sigură între aceste componente. *Aplicația Terminal* are rolul cel mai important, deoarece aceasta realizează legăturile cu celelalte componente.

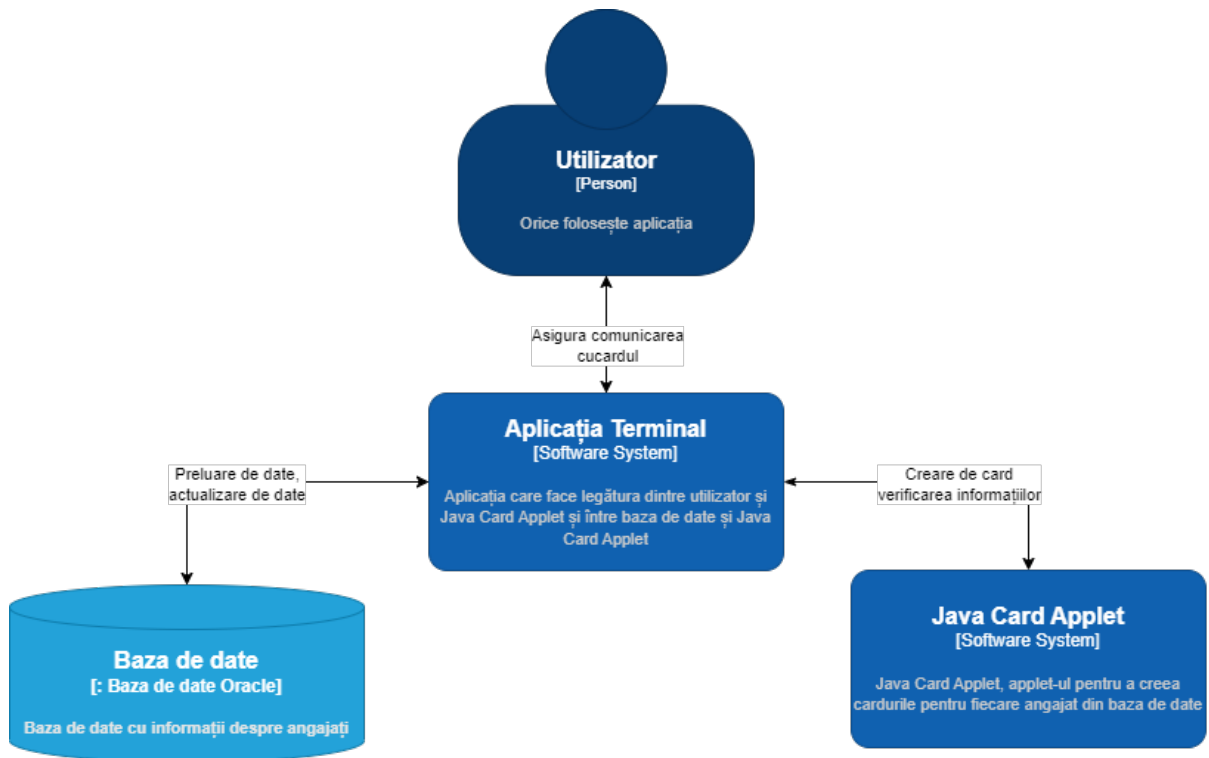


Figura 2.1: Diagrama secvențială

Principalele legături sunt următoarele:

1. Legătura între *aplicația Terminal* și baza de date. Se realizează prin intermediul clasei *DBConnection* din pachetul *dataBase* din cadrul *aplicației Terminal*. Această clasă este modelată ca un **Singleton**<sup>1</sup> și permite altor componente ale *aplicației Terminal* să obțină o conexiune la baza de date prin metoda *getConnection()*, să creeze o conexiune utilizând metoda *createConnection()* și să închidă conexiunea utilizând metoda *closeConnection()*. Astfel, această clasă asigură o gestionare adecvată a conexiunii la baza de date **Oracle**.
2. Legătura dintre *aplicația Terminal* și *Java Card Applet*, Se realizează prin intermediul clasei *Card* din pachetul *cardObjects* din cadrul *aplicației Terminal*. Clasa

<sup>1</sup>Un Singleton este un șablon de proiectare utilizat în programarea orientată pe obiecte pentru a asigura că o clasă are o singură instanță și oferă un punct global de acces la acea instanță.



*Card* se ocupă de stabilirea conexiunii cu instanța *cref* prin intermediul metodei *make\_connection()*. Instanța *cref* este inițializată cu **fișierul .cap**<sup>2</sup> al *applet*-ului din cadrul *Java Card Applet*, se creează un **socket**<sup>3</sup> către *localhost* și se obțin fluxurile de intrare și ieșire pentru comunicarea cu instanța *cref*. Aceasta este o etapă foarte importantă în gestionarea interacțiunii cu *Java Card Applet* și permite transmiterea **comenzilor APDU**<sup>4</sup> către instanța cardului pentru a fi procesate. Astfel, clasa *Card* joacă un rol crucial în asigurarea comunicării și interacțiunii eficiente între aplicația *Terminal* și *Java Card Applet*.

3. Legătura între aplicația *Terminal* și utilizator. Se realizează prin interfața grafică realizate cu **JavaFX**. Interfața grafică este definită într-un singur pachet din cadrul aplicației *Terminal*, denumit sugestiv *graphicInterface*, iar clasa responsabilă de gestionarea interfeței este denumită *GUIApplication*.

Acum că am stabilit legăturile între componentele sistemului propus, voi oferi o prezentare mai detaliată a fiecărei componente pentru a avea o înțelegere mai completă a acestora.

## 2.3 Java Card Applet

În acest subcapitol voi prezenta în detaliu componenta *Java Card Applet*, care este reprezentată de *applet*-ul din cadrul proiectului *BadgeEmployee*. Proiectul **Java Card**, *BadgeEmployee*, are o structură bine organizată, prezentată în detaliu în Figura 2.2. În continuare, voi oferi informații suplimentare despre fișierul *BadgeEmployee.java*, situat în cadrul pachetului *com.card.applet* din directorul *src*. Acest fișier reprezintă nucleul aplicației *Java Card Applet*. De asemenea, voi prezenta cele două *script*-uri importante, *cap-BadgeEmployee.script* și *BadgeEmployee.script*, localizate în directorul *apdu-scripts*. Aceste *script*-uri sunt utilizate pentru configurarea și testarea *applet*-ului descris

---

<sup>2</sup>Un fișier ".cap" din cadrul unei aplicații Java Card reprezintă un fișier în format binar care conține codul sursă, resursele și informațiile necesare pentru a încărca și executa aplicația Java Card pe o cartela inteligentă

<sup>3</sup>Un socket este o entitate de comunicare utilizată pentru a permite schimbul de date între două sau mai multe dispozitive prin rețea.

<sup>4</sup>Comanda APDU (Application Protocol Data Unit) este un format de mesaj standardizat utilizat în comunicarea între un terminal (cum ar fi un card reader) și un card inteligent (cum ar fi un Java Card Applet).

în fișierul *BadgeEmployee.java* și reprezintă un element esențial în procesul de dezvoltare și testare a aplicației **Java Card**.

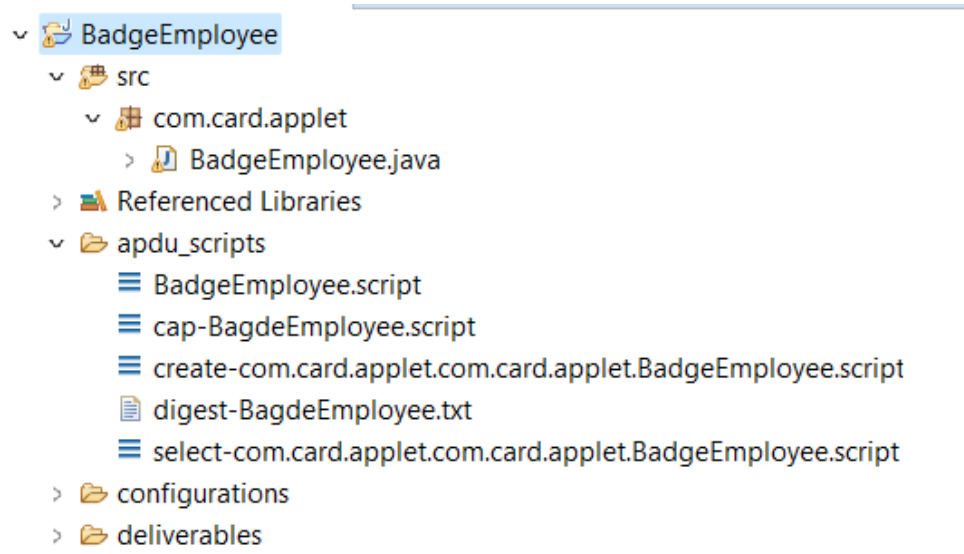


Figura 2.2: Structura proiectului BadgeEmployee

### 2.3.1 Structura applet-ului

Fisierul *BadgeEmployee.java* conține descrierea clasei *BadgeEmployee*. Această clasă extinde clasa *Applet* din cadrul **framework-ului**<sup>5</sup> **Java Card** și implementează metodele specifice unei aplicații **Java Card**.

Printre aceste metode se numără:

1. **install** : Metoda *install* este responsabilă de inițializarea și instalarea *applet*-ului pe card. Această metodă primește parametrii de instalare și informațiile necesare pentru inițializarea cardului. În cadrul acestei metode se apelează constructorul clasei *BadgeEmployee*, care va fi descris mai jos.
2. **select** : Metoda *select* este apelată atunci când *applet*-ul este selectat pe card.
3. **deselect** : Metoda *deselect* este apelată atunci când *applet*-ul este deselectat de pe card.
4. **procces** : Metoda *procces* preia și procesează comenzile primite de la un terminal, în funcție de cum a fost implementat *applet*-ul folosit. În cazul *applet*-ului creat de

---

<sup>5</sup>Un framework este o structură de dezvoltare software care oferă un set de funcționalități, biblioteci și instrumente pentru a ajuta dezvoltatorii să construiască aplicații mai rapid și mai eficient.

mine, metoda *process* va putea procesa opt comenzi prin intermediul mai multor metode, care vor fi descrise în detaliu mai jos.

Constructorul *BadgeEmployee* este declarat *private* și poate fi apelat doar din metoda *install*. Acesta primește ca parametri informațiile necesare inițializării unui card care va fi atribuit unui angajat din cadrul companiei. În cadrul constructorului din clasa *BadgeEmployee* se realizează următoarele:

1. Se inițializează variabila *pin*, care este de tipul **OwnerPIN**, cu limitele de încercări și dimensiunea maximă a codului PIN. Tot în cadrul constructorului, variabila *pin* este inițializată cu o valoare primită prin intermediul șirului de octeți *bArray* primit ca parametru.
2. Se extrage valoare ID-ului din șirul de octeți *bArray* și se atribuie această valoare variabilei *id*.
3. Se extrage numărul de zone primite și se compară cu **MAX\_ROOMS\_NUMBERS** (numărul maxim de zone). Dacă nu sunt egale, se aruncă o excepție de tip *ISOException* cu codul de eroare **SW\_WRONG\_LENGTH**. Dacă sunt egale, atunci se inițializează valorile din vectorul *acces\_areas*.
4. Se verifică dacă s-au primit fix doi octeți pentru a citi datele despre orele și minutele lucrate în ziua curentă. Dacă numărul de octeți nu este egal cu 2, atunci se aruncă o excepție *ISOException* cu codul de eroare **SW\_WRONG\_LENGTH**. Dacă numărul de octeți primiți este egal cu doi, atunci se inițializează variabilele *hours\_worked\_today* și *minutes\_worked\_today* cu cei doi octeți primiți. Înainte de inițializarea variabilei *minutes\_worked\_today*, se verifică dacă octetul primit este mai mare decât 60. Dacă este mai mare, se aruncă o excepție de tipul *ISOException* cu codul de eroare **SW\_WRONG\_DATA**. Același lucru se întâmplă și pentru variabilele *hours\_worked\_current\_month* și *minutes\_worked\_current\_month*.
5. Sunt inițializate variabilele *in\_company* și *current\_area*. Variabila *in\_company* este inițializată cu valoarea octetului *0x00*, semnificând că angajatul nu se află în companie. Variabila *current\_area* este inițializată cu valoarea octetului *0x08*, indicând că angajatul nu se află în nicio zonă.
6. La finalul constructorului se realizează un apel al funcției *register()*, care este utilizată pentru înregistrarea aplicației în sistemul **Java Card**. Apelul funcției *regis-*

*ter()* în cadrul constructorului indică că aplicația este pregătită să fie încărcată și să primească comenzi.

### 2.3.2 Comenziile suportate de applet

*Applet*-ul descris suportă opt comenzi, acestea sunt :

1. **VERIFY**. Această comandă este una de verificare a codului PIN. Este o comandă esențială deoarece celelalte comenzi descrise pot fi realizate doar dacă verificarea a avut loc cu succes. În cazul în care comanda nu s-a executat, va fi returnat un mesaj de eroare de forma **SW1, SW2**, care reprezintă un cod specific de eroare, dacă mesajul de forma **SW1, SW2** are valoare *90, 00*, atunci înseamnă că comanda a fost executată cu succes.
2. **ENTRY\_IN\_AREA**. Această comandă verifică dacă un angajat are permisiunea de a intra într-o anumită zonă, numărul zonei fiind specificat prin intermediul comenzii **APDU**. Dacă angajatul nu are permisiunea de a intra în zona respectivă, atunci va fi returnat un răspuns de forma **SW1, SW2**, care indică un cod specific de eroare.
3. **EXIT\_OF\_AREA**. Această comandă permite unui angajat să părăsească zona curentă. Dacă angajatul nu se află în nicio zonă sau nu a verificat codul PIN, atunci va fi returnat un mesaj specific de eroare.
4. **EXIT\_FROM\_COMPANY**. Această comandă permite utilizatorului să părăsească compania. Dacă comanda se execută corect, în răspuns va fi inclus și numărul de ore lucrate de angajat în ziua și luna curentă. În cazul în care comanda nu poate fi executată corect, se va răspunde cu un mesaj specific de eroare.
5. **GET\_INFORMATIONS**. Această comandă permite unui angajat să își vizualizeze informațiile stocate pe cardul său. Dacă această comandă se execută corect, va răspunde cu ID-ul angajatului, zona curentă în care se află (în cazul în care nu se află în nicio zonă se va trimite octetul *0x08*), numărul de ore lucrate în ziua și luna curentă, calculate de la ultima ieșire din companie. În caz contrar, se va returna un mesaj specific de eroare, în funcție de natura erorii.
6. **CHANGE\_PIN**. Această comandă permite angajatului să-și schimbe codul PIN. Noul cod PIN este transmis prin intermediul comenzii **APDU**, și dacă este un

cod PIN valid, atunci după executarea acestei comenzi va trebui să se verifice din nou codul PIN prin intermediul comenzii **VERIFY**, dacă se dorește accesarea altor comenzi din cadrul companiei.

7. **COMPLETE\_WORK**. Această comandă este executată imediat după ce angajatul și-a verificat codul PIN și verifică dacă acesta a acumulat cele 160 de ore de muncă pentru luna curentă. Dacă acesta a efectuat numărul de ore, va trimite octetul *0x01*, iar în caz contrar va trimite octetul *0x00*.
8. **END\_OF\_THE\_MONTH**. Această comandă se execută imediat după ce s-a executat cu succes comanda **EXIT\_FROM\_COMPANY** și verifică dacă la final de lună angajatul a acumulat cele 160 de ore de muncă. În cazul în care a acumulat numărul de ore, se va trimite octetul *0x0A*. Dacă numărul de ore lucrate este între 152 și 160, se va trimite diferența până la 160 sub formă de octet. În cazul în care numărul de ore lucrate nu este nici măcar 152, se va trimite octetul *0x09*. În cazul în care ziua curentă nu este finalul lunii, nu se va trimite niciun octet.

### 2.3.3 Metode din cadrul applet-ului

Acum că am prezentat comenzile suportate, voi vorbi puțin și despre funcțiile care gestionează comenzile **APDU** și returnează un răspuns potrivit solicitării. În acest sens, am implementat următoarele funcții:

1. Metoda *verify()*, această metodă primește ca parametru un obiect *APDU* care conține comanda **APDU** primită. În această metodă are loc verificarea codului PIN prin intermediul metodei *check()* din cadrul clasei **OwnerPIN**. Dacă PIN-ul primit prin comanda **APDU** nu este același cu PIN-ul stocat pe card, atunci se va genera un răspuns cu codul de eroare **SW\_VERIFICATION\_FAILED**. În cazul în care verificarea a fost realizată cu succes, se actualizează variabila *in\_company* cu valoarea octetului *0x01* (pentru a reține că angajatul a intrat în companie) și începe contorizarea timpului, lucru realizat cu ajutorul unui obiect *TimeDuration*.
2. Metoda *change\_pin()*, această metodă realizează schimbarea codului PIN. Primește ca parametru un obiect *APDU* în care se găsește noul cod PIN. Inițial se face o verificare dacă codul PIN a fost validat prin intermediul metodei *isValidated()* din cadrul clasei **OwnerPIN**. Dacă codul nu a fost verificat, se răspunde cu un mesaj

de eroare de tipul **SW\_PIN\_VERIFICATION\_REQUIRED**. Apoi se verifică dacă noul PIN îndeplinește proprietățile unui cod PIN (să aibă maxim 8 cifre și minim 4 cifre). Dacă noul PIN nu îndeplinește aceste cerințe, se răspunde cu un mesaj de eroare de tipul **SW\_WRONG\_LENGTH**. În cazul în care noul PIN este valid, atunci se actualizează variabila *pin*.

3. Metoda *get\_information*s, această metodă primește ca parametru un obiect *APDU*. Inițial se verifică dacă codul PIN a fost validat (același scenariu ca la metoda *change\_pin()*). Dacă verificarea codului PIN este validată, atunci se verifică dacă în răspunsul comenzii se așteaptă minim 6 octeți. Dacă această cerință nu este îndeplinită, se va returna un răspuns negativ de tipul **SW\_WRONG\_LENGTH**. Dacă numărul de octeți așteptați este în regulă, atunci se începe crearea răspunsului cu informațiile esențiale stocate pe card ale angajatului. După ce răspunsul a fost creat, acesta este trimis.
4. Metoda *entry\_in\_area()*, această metodă primește ca parametru un obiect *APDU* care conține informații legate de zona în care dorește angajatul să intre. Are loc verificarea codului PIN, iar apoi se verifică dacă s-a primit cel puțin un octet care reprezintă zona în care se dorește intrarea. Dacă această cerință nu este îndeplinită, atunci se răspunde cu un mesaj de tipul **SW\_WRONG\_LENGTH**. Dacă datele primite sunt corecte cel puțin din punct de vedere al dimensiunii, se verifică dacă octetul primit este mai mare decât numărul care reprezintă ultima cameră (în cazul sistemului meu, 7), dacă octetul primit reprezintă un număr mai mare de 7, atunci se răspunde cu un mesaj de eroare de tipul **SW\_WRONG\_DATA**. Dacă octetul primit este valid, atunci se verifică dacă angajatul respectiv are acces în zona cerută și dacă se află în altă zonă. În cazul în care angajatul nu are acces în zona cerută sau dacă este în altă zonă, atunci se va returna un mesaj de tipul **SW\_SECURITY\_STATUS\_NOT\_SATISFIED**. Dacă am ajuns până în acest punct, înseamnă că toate condițiile au fost îndeplinite și angajatul poate intra în zona respectivă, deci se actualizează variabila *current\_area* cu valoarea primită.
5. Metoda *exit\_of\_area()*, această metodă primește ca unic parametru un obiect *APDU*. În cadrul obiectului *APDU* este stocată comanda **APDU** care este satisfăcută de această funcție, comanda care permite unui angajat să părăsească zona curentă în care se află. Inițial se verifică dacă codul PIN este validat, iar după acest

test se verifică dacă angajatul se află într-o zonă și dacă este în companie. Dacă oricare dintre cele două condiții nu este satisfăcută, atunci se trimite un mesaj de eroare de forma **SW\_SECURITY\_STATUS\_NOT\_SATISFIED**. În cazul în care sunt îndeplinite ambele condiții, atunci angajatului îi este permisă ieșirea din zona curentă și variabila *current\_area* este actualizată cu octetul *0x08*.

6. Metoda *exit\_from\_company()*, Această metodă primește un singur parametru, un obiect *APDU*, și rezolvă solicitarea de ieșire din companie. Inițial se verifică dacă codul PIN a fost validat, iar după acest test se verifică dacă în răspunsul așteptat se permite transmiterea a cel puțin 4 octeți. Dacă nu se respectă această cerință, se răspunde cu un mesaj de tipul **SW\_WRONG\_LENGTH**. Dacă se permite transmiterea a cel puțin 4 octeți, atunci se oprește contorizarea timpului, se realizează un apel al metodei *update\_time()* (care va fi descrisă mai jos) și se începe formarea răspunsului care conține orele lucrate în ziua și luna curentă. După crearea răspunsului, acesta este trimis și variabila *in\_company* este setată cu octetul *0x00*.
7. Metoda *update\_time()*, această metodă nu primește niciun parametru și este folosită pentru a adăuga timpul lucrat în cadrul companiei la datele primite inițial referitoare la minutele și orele lucrate în ziua curentă, respectiv minutele și orele lucrate în luna curentă. Contorizarea timpului a fost o provocare datorită restricțiilor impuse de **Java Card** referitoare la dimensiunea datelor, dar în final am reușit acest lucru cu ajutorul unui obiect de tipul **TimeDuration**. După ce timpul petrecut în cadrul companiei a fost adăugat la timpul inițial, se verifică dacă numărul de minute este mai mare de 60. În cazul în care una dintre cele două variabile care rețin minutele lucrate depășește 60, minutele sunt transformate în ore și adăugate la variabilele care rețin numărul de ore corespunzătoare, iar restul de minute rămase sunt păstrate în variabilele care rețin minutele.
8. Metoda *check\_complete\_work*, Această metodă primește un singur parametru, un obiect *APDU*. Această funcție va verifica dacă angajatul a îndeplinit numărul de ore lucrate pe lună, adică 160. Inițial, are loc testul de verificare a codului PIN. Dacă codul este validat, se verifică dacă se așteaptă cel puțin un octet ca răspuns. În caz contrar, se va trimite un răspuns negativ de forma **SW\_WRONG\_LENGTH**. Dacă am trecut și de acest test, putem pregăti răspunsul, dar mai întâi voi verifica dacă numărul de ore de muncă din luna curentă este mai mare sau egal cu 160.

Dacă angajatul a îndeplinit acest număr de ore, se trimite octetul *0x01*. În caz contrar, se trimite octetul *0x00*.

9. Metoda *check\_end\_of\_the\_month()*, Această metodă primește un singur parametru, un obiect *APDU*, care conține informații referitoare la data curentă. Inițial, se face testul pentru verificarea codului PIN. După validarea codului PIN, se verifică dacă am primit fix 3 octeți care reprezintă data curentă. În caz contrar, răspunsul trimis va fi unul negativ de forma **SW\_WRONG\_LENGTH**. În cazul în care am primit cei 3 octeți, îi stochez într-un vector numit *current\_date* și verific dacă pot trimite în continuare cel puțin un octet. Dacă nu pot, voi răspunde cu același mesaj de eroare, **SW\_WRONG\_LENGTH**. Dacă am trecut de aceste teste, încep verificările pentru a crea răspunsul final. Mai întâi, verific dacă data primită este sfârșitul unei luni. Dacă nu respectă această cerință, nu voi trimite niciun octet înapoi. Însă, dacă ziua curentă este sfârșit de lună, verific dacă angajatul a acumulat cele 160 de ore și dacă îndeplinește această cerință, trimit octetul *0x0A*. Dacă numărul de ore este între 152 și 160, trimit un octet care reprezintă diferența dintre 160 și numărul de ore lucrate. În cazul în care numărul de ore este mai mic decât 152, trimit un octet cu valoarea *0x09*.
10. Metoda *is\_the\_last\_day()*, Această metodă primește drept parametru un șir de octeți care reprezintă o dată calendaristică, iar șirul de octeți are 3 elemente. În cadrul acestei metode, se verifică dacă data primită ca parametru reprezintă o zi din finalul unei luni. Acest lucru este realizat prin teste simple pentru a determina dacă ziua specificată este sau nu o zi de la finalul unei luni.

Acestea au fost cele mai importante aspecte ale fișierului *BadgeEmployee.java*, iar acum voi continua cu o prezentare a conceptului de comandă **APDU** și a celor două scripturi menționate mai sus.

### 2.3.4 Conceptul de comandă APDU

Comunicarea dintre card și angajat se realizează prin intermediul comenzilor **APDU**. O comandă **APDU** (*Application Protocol Data Unit*) este o unitate de date utilizată în comunicarea între un dispozitiv de card și un terminal sau o aplicație care interacționează cu cardul.



```

//55 01 = SW_ERROR_PIN_VERIFICATION
//Verify user pin
0x80 0x20 0x00 0x05 0x04 0x05 0x01 0x01 0x02 0x7F;
APDU|CLA: 80, INS: 20, P1: 00, P2: 00, Lc: 05, 04, 05, 01, 01, 02, Le: 00, SW1: 90, SW2: 00
//90 00 = SW_NO_ERROR
// Entry in area 0
0x80 0x30 0x00 0x00 0x01 0x00 0x7F;
APDU|CLA: 80, INS: 30, P1: 00, P2: 00, Lc: 01, 00, Le: 00, SW1: 90, SW2: 00
//90 00 = SW_NO_ERROR
// GET info
0x80 0x70 0x01 0x01 0x00 0xc;
APDU|CLA: 80, INS: 70, P1: 01, P2: 01, Lc: 00, Le: 06, 2a, 00, 00, 00, 00, 00, SW1: 90, SW2: 00
//90 00 = SW_NO_ERROR
// Entry in area 3
0x80 0x30 0x00 0x00 0x01 0x03 0x7F;
APDU|CLA: 80, INS: 30, P1: 00, P2: 00, Lc: 01, 03, Le: 00, SW1: 69, SW2: 82
//69 82 = SW_ERROR_NO_ACCES
// Exit of area 0
0x80 0x40 0x00 0x00 0x00 0x7F;
APDU|CLA: 80, INS: 40, P1: 00, P2: 00, Lc: 00, Le: 00, SW1: 90, SW2: 00
//69 82 = SW_ERROR_NO_ACCES

```

Figura 2.3: Schimb de comenzi APDU

## Command APDU

|     |     |    |    |    |            |    |
|-----|-----|----|----|----|------------|----|
| CLA | INS | P1 | P2 | Lc | ...Data... | Le |
|-----|-----|----|----|----|------------|----|

## Response APDU

|            |     |     |
|------------|-----|-----|
| ...Data... | SW1 | SW2 |
|------------|-----|-----|

**Body:** optional

**Header:** mandatory

Figura 2.4: Structura comenzii APDU și structura răspunsului

O comandă **APDU** este formată din două părți principale: comanda și respectiv datele asociate (opționale). Comanda include informații precum instrucțiunea solicitată și alte informații specifice operației. Comanda **APDU** este trimisă de terminal sau aplicație către card, iar cardul răspunde cu un răspuns **APDU** care conține rezultatul solicitării sau alte informații relevante.

### 2.3.5 Script-uri de comenzi APDU

Așa cum am menționat mai sus, *script*-urile *cap-BadgeEmployee.script* și *BadgeEmployee.script* au o importanță deosebită, deoarece acestea asigură configurarea cardului în cadrul sistemului și testarea cardului în cadrul dezvoltării sistemului.

*Script*-ul *cap-BadgeEmployee.script* este un fișier de configurare esențial în cadrul

aplicației **Java Card**. Acest fișier conține instrucțiuni și comenzi specifice care sunt utilizate pentru a configura și personaliza funcționalitatea cardului. *cap-BadgeEmployee.script* este deosebit de folositor și important, deoarece prin intermediul său se stabilesc setările cardului și se definește comportamentul acestuia în cadrul aplicației. Prin intermediul acestui fișier, se poate realiza testarea și verificarea corectitudinii funcționalităților cardului, asigurându-se că acesta operează conform cerințelor aplicației **Java Card**.

*Script-ul BadgeEmployee.script* este un script de comenzi **APDU** pe care l-am creat în scopul testării funcționalităților cardului în cadrul aplicației. Acest script conține o serie de comenzi **APDU** pe care le-am trimis către card pentru a verifica și valida comportamentul cardului în anumite situații.

Prin adăugarea de comenzi **APDU** specifice în cadrul *script*-ului, am putut testa opțiunile pe care cardul le poate oferi, printre care autentificarea cu codul PIN, intrarea și ieșirea din anumite zone autorizate, înregistrarea și raportarea timpului de lucru sau alte preluări de informații stocate pe card. Prin crearea și utilizarea acestui *script*, am avut posibilitatea să verific comportamentul și funcționalitățile cardului pentru a mă putea asigura că acestea sunt implementate corect.

În acest subcapitol am prezentat toate aspectele relevante și interesante legate de componenta *Java Card Applet*. Am dorit să evidențiez contribuția pe care această componentă o are asupra întregului sistem și, totodată, cum a putut rezolva o problemă aparent complicată într-un mod intuitiv și clar. Mai departe, voi prezenta celelalte două componente, *aplicația Terminal* și *baza de date*.

## 2.4 Aplicația Terminal

În acest subcapitol voi prezenta în detaliu structura, componenta și funcționalitățile pe care le oferă *Aplicația Terminal*. *Aplicația Terminal* reprezintă nucleul sistemului propus, deoarece în cadrul acesteia se realizează legăturile cu celelalte componente: *Java Card Applet*, baza de date și utilizatorul. Această aplicație este reprezentată de proiectul numit *Terminal*, un proiect **Java** care conține în total 4 pachete, care vor fi prezentate mai jos. Am ales să împart această descriere în două părți. Prima parte va descrie pachetele care fac legătura cu componentele *Java Card Applet* și baza de date, iar a doua parte va descrie pachetul care face legătura cu utilizatorul prin intermediul interfeței grafice.

## 2.4.1 Pachetele care fac legătura cu Java Card Applet si baza de date

Am să încep cu descrierea pachetului *database*, care se ocupă de crearea unei conexiuni cu baza de date și de manipularea datelor din aceasta. Acest pachet conține două clase: clasa *DBConnection* și clasa *DataBaseUtil*.

Clasa *DBConnection* este creată conform **design pattern-ului**<sup>6</sup> *Singleton* și este responsabilă de gestionarea conexiunii cu baza de date. Iată o descriere a funcțiilor din cadrul clasei:

1. Metoda statică *createConnection()*, această metodă este responsabilă de crearea conexiunii cu baza de date. Ea înregistrează *driver*-ul **Oracle JDBC**, cu scopul de a crea conexiunea.
2. Metoda statică *getConnection()*, această metodă returnează conexiunea curentă.
3. Metoda statică *closeConnection()*, această metodă are rolul de a închide conexiunea curentă cu baza de date.

În esență, clasa *DBConnection* oferă metode pentru a crea și închide conexiunea la baza de date, astfel încât cealaltă clasă din pachetul *database* să poată manipula datele în mod corespunzător.

Clasa *DatabaseUtil* are ca scop principal manipularea datelor din baza de date. Prin intermediul acestei clase și a metodelor implementate în cadrul ei, am realizat operații asupra bazei de date, cum ar fi selectări, inserări și actualizări. Constructorul clasei primește ca parametru un obiect *DateProvider* (această clasă va fi descrisă mai jos, în cadrul pachetului *utilObjects*), prin intermediul căruia se obține o dată calendaristică, care poate fi data curentă sau o dată la alegere. În cadrul acestei clase am următoarele metode:

1. Metoda *getNameFromDB()*, această metodă primește un singur parametru, un obiect de tip *Integer* care reprezintă ID-ul unui angajat. Metoda returnează un șir de caractere, *String*, reprezentând numele angajatului cu ID-ul specificat ca parametru.
2. Metoda *getHoursWorkedTodayFromDB()*, această metodă primește un singur parametru, un obiect de tip *Integer* care reprezintă ID-ul unui angajat. Această metodă returnează un vector de două obiecte de tip *Integer*, reprezentând numărul

---

<sup>6</sup>model de proiectare

de ore și minute lucrate de către angajatul cu ID-ul specificat ca parametru, în ziua stabilită în cadrul constructorului clasei *DataBaseUtil*.

3. Metoda *getHoursWorkedCurrentMonthFromDB()*, această metodă primește un singur parametru, un obiect de tip *Integer* care reprezintă ID-ul unui angajat. Această metodă returnează un număr întreg care reprezintă numărul de ore lucrate de către angajatul cu ID-ul specificat ca parametru, în cadrul lunii din data primită în constructorul clasei *UtilDateBase*. În cadrul acestei metode, sunt luate în considerare minutele lucrate în fiecare zi și sunt convertite în ore.
4. Metoda *getAreasAccesFromDB*, această metodă primește un singur parametru, un obiect de tip *Integer* care reprezintă ID-ul unui angajat. Această funcție returnează un vector de elemente de tip *Boolean*, care reprezintă dacă angajatul cu ID-ul specificat ca parametru are acces în zona *x*. Dacă valoarea elementului de la poziția *x* din vectorul returnat este *True*, înseamnă că angajatul are acces în zona respectivă. În cazul în care valoarea este *False*, semnifică că angajatul nu are acces în zona *x*.
5. Metoda *getPinFromDB()*, această metodă primește un singur parametru, un obiect de tip *Integer* care reprezintă ID-ul unui angajat. Această funcție returnează un șir de caractere , *String*, care reprezintă codul PIN al angajatului cu ID-ul specificat ca parametru.
6. Metoda *insertNewDate()*, această metodă primește un singur parametru, un obiect de tip *Integer* care reprezintă ID-ul unui angajat. Această metodă este declarată de tipul *void* deoarece în cadrul ei se realizează o inserție în baza de date, adică se adaugă o înregistrare nouă care nu există până în momentul testării în baza de date. Această înregistrare reprezintă o zi de lucru a unui angajat cu ID-ul specificat ca parametru.
7. Metoda *isDateAlreadyExists()* , această metodă primește un singur parametru, un obiect de tip *Integer* care reprezintă ID-ul unui angajat. Această funcție verifică dacă există în baza de date o înregistrare a unui angajat cu ID-ul specificat ca parametru, în data setată prin intermediul constructorului.
8. Metoda *updateNewDate()* , această metodă primește trei parametri, trei obiect de tip *Integer* care reprezintă ID-ul unui angajat, numărul de ore și minute lucrate

actualizate. Această metodă actualizează orele și minutele lucrate în ziua specificată ca parametru în constructor, pentru angajatul cu ID-ul specificat în parametrii funcției.

9. Metoda *getEmployeesId()*, această metodă nu primește niciun parametru. Ea returnează un *ArrayList* de obiecte *Integer* care reprezintă ID-urile fiecărui angajat din baza de date.
10. Metoda *updatePIN*, această metodă primește doi parametri, un obiect de tip *Integer* care reprezintă ID-ul unui angajat și un șir de caractere, *String*, care reprezintă noul cod PIN. Metoda actualizează codul PIN al angajatului cu ID-ul specificat în parametrii funcției.

Acestea au fost cele două clase din cadrul pachetului *dataBase*. Acum voi continua cu descrierea celorlalte două pachete, *cardObjects* și *utilObjects*.

Acum voi detalia singura clasă din cadrul pachetului *cardObjects*, și anume clasa *Card*. În cadrul constructorului acestei clase, se realizează următoarele acțiuni: se pornește o instanță *cref.bat* care simulează un card pentru un angajat, se stabilește o conexiune între procesul care menține instanța *cref.bat* deschisă și aplicația *Terminal*. Aceasta se realizează prin intermediul unui obiect **CadDevice** prin **protocolul T1**<sup>7</sup>. De asemenea, se inițiază comunicarea între card și aplicația *Terminal*, iar instanța *cref.bat* este instanțiată cu fișierul *BadgeEmployee.script* pentru a beneficia de funcționalitățile *applet*-ului *BadgeEmployee*. Această clasă se ocupă de comunicarea cu componenta *Java Card Applet*. Iată câteva funcții importante din cadrul acestei clase:

1. metoda *toAPDU()*, această metodă primește un singur parametru, un șir de caractere, un obiect de tipul *String*, care reprezintă o comandă **APDU**. Scopul acestei metode este de a converti comanda primită într-un șir de octeți pentru a putea fi trimisă către card. Conversia de la *String* la un șir de octeți se realizează prin parsarea comenzii primite și prin efectuarea unor operații simple de convertire.
2. Metoda *executeCommand*, această metodă primește un singur parametru, reprezentat de un șir de caractere, un obiect de tipul *String*. Metoda returnează un obiect de tipul *Apdu*, care reprezintă răspunsul cardului la comanda primită în

---

<sup>7</sup>Protocolul T1 este un standard de comunicație utilizat în cardurile inteligente, care permite transmiterea și recepționarea datelor între card și terminal în mod asincron și controlează interacțiunea între cele două dispozitive prin intermediul unui set de reguli și comenzi specifice.

format *String* din parametrul funcției. În cadrul acestei metode se realizează schimbul de comenzi **APDU** între terminal și card prin intermediul metodei *exchangeAdu()* din cadrul clasei **CadClientInterface**.

3. Metoda *install\_cap\_files()*, această metodă este declarată ca fiind de tipul *void* și nu primește niciun parametru. În cadrul acestei metode voi parse fișierul *cmds\_apdu.txt*, care conține comenzile de inițializare a cadrului, adică conține comenzile din fișierul *cap-BadgeEmployee.script*. După ce voi parse fișierul, voi converti fiecare comandă de la tipul *String* la un șir de octeți prin intermediul funcției *toAdu()*, iar apoi voi executa fiecare șir de octeți utilizând metoda *exchangeAdu()* din cadrul clasei **CadClientInterface**.

Acestea sunt aspectele cheie ale clasei *Card* din pachetul *cardObjects*. În cadrul acestei clase, am stabilit conexiunea cu cardul utilizând diferite clase și metode disponibile în pachetul *com.sun.javacard.apduio*. În continuare, voi descrie ultimul pachet numit *utilObjects*, care se ocupă de legătura între baza de date, *Java Card Applet* și aplicația *Terminal*.

În cadrul pachetului *utilObjects*, am implementat trei clase cu scopul de a facilita gestionarea și structurarea legăturilor dintre componentele aplicației și de a crea un management mai eficient al datelor. Aceste clase contribuie la o organizare mai clară și coerentă a funcționalităților. Pachetul *utilObjects* cuprinde următoarele clase: *DateProvider*, care se ocupă de stocarea datelor, *Employee*, care gestionează informațiile despre angajați, și *Usecase*, care definește cazurile de utilizare specifice aplicației, adică se ocupă cu crearea de comenzi **APDU** în format *String*.

**Clasa *DateProvider*.** În cadrul acestei clase, se va memora data la care se dorește efectuarea operațiilor cu cardul. Clasa *DateProvider* oferă posibilitatea de a seta o dată curentă sau o dată la alegere. Constructorul clasei stochează în variabila membru *currentDate* data curentă sub forma *ZZ-LL-AAAA*. De asemenea, în clasa se găsesc două funcții care au rolul de a prelua informații din cadrul clasei. Prima funcție, *getDate()*, returnează data în format *String*, iar a doua funcție, *getCurrentMonth()*, returnează un număr întreg reprezentând luna din data setată în cadrul clasei. O funcție cu un impact mai important este *setCustomDate()*, care primește trei parametri, reprezentând ziua, luna și anul datei pe care dorim să o setăm. În cadrul acestei funcții are loc schimbarea datei curente cu o dată la alegere

**Clasa *Employee*.** În cadrul acestei clase are loc preluarea datelor despre un anumit

angajat și stocarea acestora la nivelul clasei. Constructorul clasei primește doi parametri: un număr întreg care reprezintă ID-ul unui angajat și un obiect *DataProvider* care reprezintă data curentă. În cadrul constructorului au loc interacțiuni cu baza de date pentru a prelua informațiile despre angajatul cu ID-ul specificat din parametrii constructorului. Aceste operații cu baza de date se realizează prin intermediul unui obiect *DatabaseUtil* descris anterior. În rest, în cadrul clasei se găsesc metode pentru a accesa informațiile despre angajat, cum ar fi codul PIN, numărul de ore și minute lucrate, ID-ul și permisiunile de acces în diferite zone ale companiei.

Clasa *Usecase*. Scopul principal al acestei clase este de a asigura modalitatea de comunicare între *aplicația Terminal* și *Java Card Applet*, mai precis, această clasă se ocupă de crearea de comenzi **APDU**. În cadrul acestei clase, am descris metode pentru a crea comenzi specifice și suportate de *Java Card Applet*, cum ar fi următoarele :

1. Metoda *createCommandForCreateApplet()*, această metodă generează comanda pentru crearea unui card destinat unui angajat. Comanda este returnată în format *String*.
2. Metoda *createCommandForCheckPin*, această metodă generează comanda de verificare a codului PIN. Rezultatul acestei metode va fi returnat sub forma unui șir de caractere, *String*.
3. Metoda *createCommandForCheckLastDay*, această metodă creează comanda pentru a verifica dacă angajatul se află în ultima zi a lunii și dacă a îndeplinit numărul necesar de ore lucrate. Rezultatul acestei comenzi va fi returnat sub formă de obiect *String*.
4. Metoda *createCommandForEntryInArea*, această metodă creează comanda responsabilă cu intrarea unui angajat într-o anumită zonă. Metoda va returna comanda sub forma unui șir de caractere.
5. Metoda *createCommandForChangePin*, Această metodă returnează comanda specifică pentru operațiunea de schimbare a codului PIN, sub forma unui șir de caractere.

Acestea au fost pachetele din cadrul proiectului *Terminal* care se ocupă de legătura între *aplicația Terminal*, *Java Card Applet* și *baza de date*. În continuare, voi prezenta pachetul care se ocupă de legătura dintre *aplicația Terminal* și utilizator, adică pachetul responsabil de realizarea interfeței grafice.

## 2.4.2 Pachetul care face legătura cu utilizatorul

Pachetul responsabil de realizarea acestei legături este pachetul *graphicInterface*. Acest pachet cuprinde un ansamblu de clase care, împreună, construiesc interfața grafică a aplicației *Terminal*. Pentru a beneficia de avantajele dezvoltării interfeței grafice, am decis să folosesc **framework-ul JavaFX**, deoarece acesta oferă tot ce am avut nevoie și dispune de un set complex de instrumente *software*. În cadrul pachetului *graphicInterface*, am descris următoarele clase:

Clasa *GUIApplication*. Aceasta este clasa principală a interfeței grafice, deoarece se ocupă de gestionarea celorlalte clase din pachetul *graphicInterface*. Pe lângă aceasta, clasa se ocupă și de proiectarea aspectului și dimensiunilor ferestrei grafice.

Clasa *SelectDateMeniu*. Această clasă se ocupă de crearea paginii de selectare a datei. Prin intermediul ei, utilizatorul poate selecta data dorită, fie că este data curentă sau o dată la alegere. Dacă utilizatorul dorește să introducă o dată la alegere și introduce o dată incorectă din punct de vedere calendaristic, atunci se va afișa un mesaj de eroare. Vezi figura 2.5.

Clasa *EmployeesMeniu*. Această clasă creează o pagină cu toți angajații din baza de date. Scopul acestei pagini este de a selecta angajatul pe care dorim să-l utilizăm în cadrul aplicației.

Clasa *CreateUsecasePage*. Această clasă creează o pagină în care sunt prezentate comenzile de creare și selecție ale cardului pentru angajatul pe care l-am selectat anterior. Dacă una dintre cele două comenzi nu s-a executat corect, se va afișa un mesaj de eroare și nu se va permite continuarea în cadrul aplicației. Vezi figura 2.6

Clasa *EntryInCompanyMeniu*, Această clasă creează pagina pentru autentificarea în cadrul companiei. Pentru a intra în companie, angajatul trebuie să introducă codul PIN. Dacă introduce de 3 ori consecutiv un cod PIN invalid, accesul său în cadrul companiei va fi restricționat. Vezi figura 2.7.

Clasa *InCompanyMeniu*, Aceasta creează o pagină care afișează toate funcționalitățile disponibile pentru un angajat care se află în cadrul companiei și a trecut de verificarea codului PIN. Pagina cuprinde următoarele funcționalități: Intrarea într-o anumită zonă, ieșirea din zona curentă, afișarea informațiilor stocate pe cardul său, schimbarea codului PIN și ieșirea din companie. Fiecare funcționalitate poate genera o eroare dacă nu este accesată corect, cum ar fi următoarele cazuri:

1. Intrarea angajatului într-o zonă în care nu are permisiunea.



2. Schimbarea codului PIN cu un cod invalid.
3. Ieșirea dintr-o zonă în care acesta nu se află.
4. Neverificarea codului PIN după schimbarea acestuia.
5. Intrarea într-o zonă în timp ce angajatul se află deja în altă zonă.
6. Orice eroare care apare în executarea unei comenzi **APDU**.

Aceasta a fost prezentarea aplicației Terminal. Prin aceasta prezentare am dorit să evidențiez rolul aplicației Terminal și structura sa complexă. În continuare, voi prezenta baza de date utilizată în cadrul aplicației.



The screenshot shows a web browser window titled "Employee page". The main heading is "Alege data". Below it are two buttons: "Data curentă" and "Data personalizată". Under "Data personalizată", there are three input fields containing the values "2023", "03", and "32". Below these fields is a "Submit" button. At the bottom, a red error message states "Data introdusă nu este corectă."

Figura 2.5: Pagina de selectare a datei.

Employee page

## Cardul de acces al lui Bordei Mihai

Acesta este cardul tau

Bordei Mihai

Raspunsul la comanda de creare a cardului :

CLA: 80, INS: b8, P1: 00, P2: 00, Lc: 2a, 0f, 0a, 00, 00, 00, 00, 0a, 00, 02, 03, 04

Raspunsul la comanda de selectare a cardului :

CLA: 00, INS: a4, P1: 04, P2: 00, Lc: 0f, 0a, 00, 00, 00, 00, 0a, 00, 02, 03, 04, 05

Inapoi

Întrare in companie

Figura 2.6: Pagina de creare și selectare a cardului.

Employee page

## Tasteaza codul PIN

12345

Submit

CLA: 80, INS: 20, P1: 00, P2: 00, Lc: 05, 01, 02, 03, 04, 05, Le: 00, SW1: 90, SW2: 00

Bine ai venit ! < Bordei Mihai 44:04:20:06:2023 >

CLA: 80, INS: 90, P1: 00, P2: 00, Lc: 00, Le: 01, 00, SW1: 90, SW2: 00

Inca nu ai acumulat cele 160 de ore

Inapoi

In companie

Figura 2.7: Pagina de verificare a codului PIN.

## 2.5 Baza de date

Baza de date este o componentă importantă în cadrul aplicației mele, deoarece oferă un sistem adecvat de stocare a informațiilor despre angajați și zilele de lucru ale acestora. Pentru implementarea bazei de date în aplicația mea, am decis să utilizez o bază de tipul **Oracle**, un sistem de gestiune a bazelor de date fiabil și intuitiv. Alegerea bazei de date de tipul **Oracle** a fost determinată de funcționalitățile sale avansate, securitatea riguroasă și performanța înaltă pe care le oferă. Această bază de date îmi permite să organizez și să accesez eficient datele angajaților și informațiile legate de programul lor de lucru.

Baza de date dezvoltată de mine este formată din două tabele: *employee* și *worked\_days*. Aceste tabele sunt legate printr-o relație de tipul *one to many*, ceea ce înseamnă că un angajat poate avea mai multe zile lucrate, în timp ce o zi lucrată corespunde unui singur angajat. Relația *one to many* este implementată prin intermediul unei chei străine în tabela *worked\_days*, care face referire la cheia primară a tabelului *employee*. Aceasta asigură integritatea referențială și permite asocierea corectă a zilelor lucrate cu angajații corespunzători în baza de date. Aceste două tabele au fost descrise în diagrama ERD<sup>8</sup> prezentată în figura 2.8.

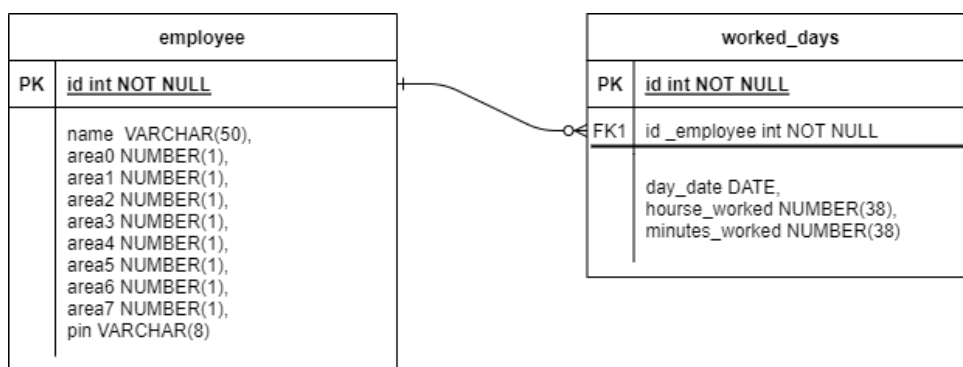


Figura 2.8: Diagrama ERD a bazei de date.

Pentru a putea testa funcționalitățile sistemului propus, am populat baza de date cu un număr favorabil de angajați și un număr relativ mic de zile lucrate. Popularea tabelului *worked\_days* a fost realizată pentru luna martie a anului 2023, astfel încât să acopere toate scenariile de utilizare posibile. Pentru a facilita partea de populare și inserare a datelor în baza de date de către *aplicației Terminal*, am creat două **trigger**-

<sup>8</sup>Diagrama ERD (Entity-Relationship Diagram) este o reprezentare grafică a structurii unei baze de date.

uri<sup>9</sup>: *trg\_generate\_id\_employee* și *trg\_generate\_id\_worked\_days*, care alocă un ID unic pentru fiecare înregistrare inserată în fiecare tabelă.

| ID | NAME            | AREA0 | AREA1 | AREA2 | AREA3 | AREA4 | AREA5 | AREA6 | AREA7 | PIN   |
|----|-----------------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| 61 | Trifan Andrei   | 0     | 0     | 0     | 0     | 1     | 1     | 1     | 1     | 00000 |
| 63 | Rusu Raluca     | 0     | 1     | 0     | 1     | 1     | 1     | 1     | 1     | 00000 |
| 64 | Codreanu Andrei | 0     | 1     | 0     | 1     | 1     | 1     | 1     | 1     | 00000 |
| 41 | Bordei Mihai    | 0     | 0     | 1     | 1     | 0     | 0     | 1     | 1     | 12345 |
| 42 | Popa Mihai      | 1     | 0     | 0     | 1     | 1     | 1     | 0     | 0     | 54321 |

Figura 2.9: Înregistrările din tabela employees.

## 2.6 Probleme întâmpinate și soluțiile acestora

În procesul dezvoltării acestui sistem, am întâlnit diverse provocări care m-au captivat și m-au motivat să găsesc soluții concise și sigure. În cele ce urmează, voi prezenta câteva dintre aceste probleme semnificative care au fost în cele din urmă rezolvate.

Prima provocare pe care am întâmpinat-o în dezvoltarea aplicației a fost contorizarea timpului de lucru al unui angajat. În cadrul aplicației *Java Card Applet*, am constatat că restricțiile de dimensiune ale datelor în mediul **Java Card** nu îmi permiteau să utilizez metodele obișnuite disponibile în **Java**. După o cercetare mai aprofundată în documentația oferită de **Oracle**, am descoperit o clasă care părea să rezolve această problemă, și anume clasa *TimeDuration* din pachetul *import javacardx.framework.time*. Cu ajutorul metodelor din cadrul clasei *TimeDuration*, precum și prin utilizarea valorii returnate de metoda *SysTime.uptime()*, am reușit să calculez timpul de lucru acumulat de un angajat în cadrul companiei. Folosind aceste metode, am putut înregistra momentul începerii și terminării lucrului și să determin durată exactă a timpului de lucru al angajatului.

O altă provocare cu care m-am confruntat a fost accesarea unui card în cadrul aplicației *Terminal*. Aveam nevoie de o modalitate prin care să pot trimite comenzi **APDU** către *Java Card Applet* din aplicația *Terminal*. După o analiză mai detaliată a problemei și o cercetare amănunțită, am descoperit o soluție simplă și intuitivă în documentația oferită de **Oracle** despre aplicațiile **Java Card**. Am utilizat o instanță *cref.bat* pentru a avea un emulator în care să ruleze *applet*-ul din cadrul componen-

<sup>9</sup>Un trigger în cadrul unei baze de date este un obiect care reacționează automat la anumite evenimente sau acțiuni care au loc asupra datelor din baza de date

tei *Java Card Applet*. L-am inițializat cu fișierul *.cap* corespunzător *applet*-ului și apoi am descoperit o cale de comunicare între acesta și aplicația Terminal, prin intermediul unui obiect de tipul *CadClientInterface* din pachetul *com.sun.javacard.apduio.CadDevice*. Acest obiect mi-a oferit posibilitatea de a inițializa instanța *cref.bat* cu fișierul *.cap* specific *applet*-ului utilizat și de a schimba comenzi **APDU** între aplicația Terminal și card.

O altă problemă pe care am întâmpinat în cadrul dezvoltării aplicației a fost găsirea unei modalități de ilustrare a funcționalităților oferite de combinația dintre *Aplicația Terminal*, *Java Card Applet* și baza de date. Am considerat că cea mai potrivită abordare ar fi crearea unei interfețe grafice care să prezinte într-un mod clar și intuitiv funcționalitățile și aspectele sistemului propus. Această soluție a generat o altă problemă, și anume alegerea cea mai potrivită și bună pentru a crea interfața grafică.

După câteva zile de documentare în privința acestui aspect, am ajuns la concluzia că cea mai bună alegere pentru realizarea interfeței grafice ar fi **JavaFX**. *Framework*-ul **JavaFX** mi-a oferit toate uneltele *software* necesare pentru realizarea unei interfețe grafice moderne, ușor de înțeles și cu un aspect plăcut.

## Capitolul 3

# Funcționalități și avantaje ale cardurilor inteligente

În cadrul acestui capitol, voi prezenta funcționalitățile și avantajele cardurilor inteligente. Mă voi concentra în special pe două funcționalități importante pe care un card inteligent le poate oferi, și anume pontajul orelor de muncă și accesul în diferite zone. Am ales să mă axez pe aceste funcționalități deoarece sunt implementate în sistemul propus de mine și pot fi explicat în detaliu în acest context.

În ceea ce privește avantajele utilizării cardurilor inteligente, voi prezenta o perspectivă generală asupra beneficiilor pe care acestea le aduc într-un mediu organizațional. De asemenea, voi evidenția beneficiile specifice pe care le-am observat în aplicațiile create de mine și cum acestea au contribuit la îmbunătățirea performanței sistemului.

### 3.1 Funcționalități

Cardurile inteligente oferă funcționalități avansate de stocare și procesare a datelor. Ele pot fi utilizate pentru autentificare și acces securizat în sisteme și servicii. Pentru a oferi o înțelegere mai clară asupra funcționalităților unui card inteligent, voi descrie câteva dintre funcționalitățile pe care le-am implementat în cadrul sistemului meu.

#### 3.1.1 Accesul în diferite zone

Prin intermediul unui card inteligent, în cadrul unei companii, se poate implementa un sistem avansat de control al accesului. Acesta permite gestionarea și mo-

monitorizarea intrării angajaților în diferite zone sau încăperi cu restricții de securitate. Beneficiile acestei funcționalități sunt multiple atât pentru angajați, cât și pentru companie.

Un exemplu concret de utilizare a acestei funcționalități este integrarea cardurilor inteligente în aplicația dezvoltată de mine. Prin intermediul cardurilor inteligente, angajații au posibilitatea de a accesa anumite zone restricționate doar dacă au permisiunea de a intra în zonele respective. Astfel, se asigură o securitate sporită și se previne accesul neautorizat în acele zone.

Implementarea acestei funcționalități a fost relativ simplă. În cadrul aplicației, am creat o bază de date care stochează informații despre fiecare angajat și permisiunile de acces pe care le au. Cardurile inteligente sunt asociate cu fiecare angajat și sunt programate să conțină aceste permisiuni de acces.

Un scenariu posibil ar putea fi, atunci când un angajat dorește să intre într-o zonă restricționată, acesta trebuie să atingă cardul de cititorul specific din apropierea ușii. Cardul este citit și informațiile sunt trimise către aplicație pentru a fi verificate. Aplicația compară informațiile de pe card cu baza de date și decide dacă angajatul are permisiunea de acces în zona respectivă. În funcție de rezultat, ușa se va deschide sau va rămâne blocată.

### **3.1.2 Pontajul orelor de lucru**

Cardurile inteligente oferă o soluție eficientă și precisă pentru pontajul orelor de lucru în cadrul unei companii. Prin intermediul acestor carduri, angajații pot înregistra cu ușurință intrările și ieșirile din locul de muncă, oferind astfel un sistem automatizat de monitorizare a timpului lucrat. Angajatul trebuie doar să apropie cardul de dispozitivul de pontaj pentru a înregistra intrarea sau ieșirea din locul de muncă. Informațiile despre timpul de intrare și ieșire sunt înregistrate automat în sistemul centralizat, eliminând necesitatea înregistrărilor manuale sau a folosirii altor metode tradiționale de pontaj.

Pentru a ilustra mai bine beneficiile aduse de cardurile inteligente în procesul de pontaj al orelor de muncă, voi folosi exemplul aplicației create de mine, care implementează această funcționalitate prin intermediul unui card inteligent. Cardul inteligent începe monitorizarea timpului de lucru odată ce angajatul a intrat în companie și se oprește din monitorizare la ieșirea din companie, înregistrând informațiile despre tim-

pul lucrat și trimițându-le către baza de date pentru a fi stocate. Această comunicare între card și baza de date se realizează prin intermediul *aplicației Terminal*.

Astfel, exemplul aplicației create de mine evidențiază modul în care cardurile inteligente și aplicațiile asociate facilitează procesul de pontaj al orelor de muncă, aducând beneficii atât angajaților, cât și companiei prin automatizarea și simplificarea acestui proces important.

## 3.2 Avantaje

Cardurile inteligente oferă numeroase avantaje prin utilizarea lor în diferite domenii și aplicații. Iată o descriere a unor avantaje ale cardurilor inteligente :

1. *Portabilitate și ușurință în utilizare:* Cardurile inteligente sunt compacte și ușor de transportat, ceea ce le face convenabile pentru utilizatori. Acestea pot fi purtate în portofele sau integrate în dispozitive mobile sau alte obiecte personale. Utilizatorii pot beneficia de funcționalitățile cardului inteligent în orice moment și în orice loc, fără a avea nevoie de documente suplimentare sau dispozitive complexe.
2. *Stocare și procesare eficientă a datelor:* Cardurile inteligente dispun de capacități de stocare și procesare a datelor, permițând utilizatorilor să aibă acces rapid la informații și să efectueze diverse operații. Acestea pot fi programate pentru a realiza verificarea codurilor PIN, semnarea digitală a documentelor sau procesarea tranzacțiilor financiare.
3. *Autentificare și identificare precisă:* Cardurile inteligente sunt utilizate pentru autentificare și identificare precisă a utilizatorilor în diferite sisteme și servicii. Ele pot conține informații personale relevante, cum ar fi date de identificare, permisiuni de acces sau privilegii specifice. Utilizarea cardurilor inteligente pentru autentificare și identificare contribuie la creșterea nivelului de siguranță și la reducerea riscului de fraude sau utilizare neautorizată a datelor personale.
4. *Reducerea costurilor și creșterea eficienței:* Utilizarea cardurilor inteligente poate aduce beneficii economice semnificative. Acestea pot reduce costurile asociate cu gestionarea și administrarea sistemelor de acces și de pontaj tradiționale. De exemplu, elimină nevoia de înlocuire periodică a cheilor fizice sau a altor dispozitive de acces, reducând costurile de întreținere.



# Concluzii

Prin această lucrare am vrut să arăt cât de utile și folositoare pot fi cardurile inteligente, în cadrul mai multor domenii. Pentru a evidenția pe deplin acest fapt, am creat și o aplicație care se bazează pe un card inteligent. În cadrul sistemului implementat de mine, am încercat să simulez folosirea unui card inteligent în cadrul unei companii, pentru a putea evidenția simplitatea implementării unui astfel de sistem în diferite situații practice. Pentru a evidenția beneficiile aduse de utilizarea cardurilor inteligente, în cadrul implementării mele, am adăugat mai multe funcționalități. Am considerat că este important să includ în cardul inteligent următoarele funcționalități: autentificare bazată pe verificarea unui cod PIN, accesul în anumite zone cu restricții, înregistrarea orelor de lucru, verificarea finalizării numărului de ore de lucru pe luna curentă și verificarea finalizării numărului de ore lucrate până la sfârșitul lunii curente, precum și posibilitatea de schimbare a codului PIN. Aceste funcționalități au avut scopul de a îmbunătăți eficiența și securitatea utilizării cardului în cadrul unei companii, demonstrând astfel cât de util și versatil poate fi un card inteligent.

De asemenea, prin prezentarea argumentelor și explicațiilor corespunzătoare, am putut sublinia faptul că cardurile inteligente pot contribui la sporirea eficienței în cadrul anumitor domenii. Aceste dispozitive oferă un nivel ridicat de protecție prin autentificare sigură și restricționarea accesului în zone sensibile, asigurând astfel că doar personalul autorizat are permisiunea de a intra în aceste locații.

Prin urmare, pot afirma cu încredere că aceste carduri inteligente au capacitatea de a îmbunătăți semnificativ procesele și operațiunile, ducând la eficiență sporită, control mai bun și securitate crescută.

# Bibliografie

- Cătălin Bîrjoveanu, *Smart Cards and Applications* - 2023, [link](#)
- Frasinaru Cristian, *Advanced Programming* - 2023, [link](#)
- Oracle, *Database Documentation*, [link](#)
- Oracle, *JavaFX Documention*, [link](#)
- Oracle, *Java Card Development Kit User Guide*, [link](#)
- Oracle, *Java Card 3.1 Documentation*, [link](#)
- Oracle, *The APDU I/O API*, [link](#)
- Oracle, *TimeDuration (Java Card API, Classic Edition)*, [link](#)
- Oracle, *The Java Tutorials*, [link](#)
- Scott B. Guthery, Timothy M. Jurgensen, *Smart Card Developer's Kit* Macmillan Technical Pub., 1998, [link](#)
- Wolfgang Rankl , Wolfgang Effing, *Smart Card Handbook* Fourth Edition, John Wiley & Sons, 2010, [link](#)